```
% Stochastic Block Model for the Mitochondrial Network
% Input: N/A
% Output: An adjacency matrix for a graph with an expected community
% structure and edge density as the brain network.
% Other functions used: numedges.m
% Last modified May 29, 2015

function Adjacency_Mat = StochasticBlockModelMitochondria()
    % We start by specifying the number of nodes and communities.

    Nodes_num = 747;
    groupnum = 12;

    % Given the number of groups is groupnum, edge_probs is a list of
    % length groupnum containing the probability any two edges in a given
    % community are connected. Omega_prob is the probability two nodes
not
    % in the same community are connected.

    edge_probs = [0.0129296, 0.0235638, 0.0235759, 0.125,
0.197861,0.0904762, 0.380952,  0.266667, 0.5, 0.2,  0.2, 0.25];
    omega_prob = 0.00126098;

    % The comm_probs is a list of length groupnum+1 whose first and last
    % entries are 0 and 1 respectively. Entry i+1 is the probability of
    % being in one of the communities from 1 to i. I.e., it is a
    % cummulative probability.

    comm_probs = [0., 0.392321, 0.647746, 0.781302, 0.863105,
0.919866,0.944908, 0.956594, 0.966611, 0.976628, 0.984975, 0.993322, 1.];

    % The following functions are defined to be inputted into the GPU for
    % the purposes of speed.

    function y = edge_assign(x,prob);
        % This function is a simple true/false returner for parsing to
the
        % arrayfun and handled by the GPU.
        if x > 1- prob
            y = 1;
        else
            y = 0;
        end
    end

    function y = group_assign(x);
        % Given the cumulative probability array, comm_probs, we
        % iteratively determine which value range, between 0 and 1, the
        % random number (also between 0 and 1) belongs to and returns the
        % index.
        y = 0;
        for i = 2:(groupnum+1)
            if x < comm_probs(i)
                y = i-1;
```

```
                break
            end
        end
    end

    function prob = prob_assign(g1,g2);
        % We input g1 and g2, which are the groups that two potentially
        % adjacent nodes belong to and return the probability from the
        % array edge_probs.
        if g1==g2
            prob = edge_probs(g1);
        else
            prob = omega_prob;
        end
    end

    % Groups designation is the matrix whose i-th entry is the community
    % number the i-th node belongs to.
    % Prob_mat is the matrix of probability values, where the (i,j)-th
    % entry contains the probability node i and j are connected.
    % M is an upper triangular matrix whose entries are random numbers
from
    % 0 to 1.
    % Lastly, Adjacency_Mat is the functions output, which is the
adjacency
    % matrix for the resulting Stochastic Block Model.

    Groups_designation =
arrayfun(@group_assign,gpuArray.rand(Nodes_num,1));
    Prob_mat = arrayfun(@prob_assign,
Groups_designation,Groups_designation');
    M = triu(gpuArray.rand(Nodes_num,Nodes_num),1);
    Adjacency_Mat = arrayfun(@edge_assign,(M + M'),Prob_mat);

end
```

```
% Stochastic Block Model for the Brain Network
% Input: N/A
% Output: An adjacency matrix for a graph with an expected community
% structure and edge density as the brain network.
% Other functions used: numedges.m
% Last modified May 29, 2015

function Adjacency_Mat = StochasticBlockModelBrain()
    % We start by specifying the number of nodes and communities.

    Nodes_num = 213;
    groupnum = 3;

    % Given the number of groups is groupnum, edge_probs is a list of
    % length groupnum containing the probability any two edges in a given
    % community are connected. Omega_prob is the probability two nodes
not
    % in the same community are connected.

    edge_probs = [0.2769,0.2966,0.4717];
    omega_prob = 0.0133;

    % The comm_probs is a list of length groupnum+1 whose first and last
    % entries are 0 and 1 respectively. Entry i+1 is the probability of
    % being in one of the communities from 1 to i. I.e., it is a
    % cummulative probability.

    comm_probs = [0., 0.446009, 0.816901, 1.];

    % The following functions are defined to be inputted into the GPU for
    % the purposes of speed.

    function y = edge_assign(x,prob);
        % This function is a simple true/false returner for parsing to
the
        % arrayfun and handled by the GPU.
        if x > 1- prob
            y = 1;
        else
            y = 0;
        end
    end

    function y = group_assign(x);
        % Given the cumulative probability array, comm_probs, we
        % iteratively determine which value range, between 0 and 1, the
        % random number (also between 0 and 1) belongs to and returns the
        % index. This determines whether a certain node belongs to a
        % certain community.
        y = 0;
        for i = 2:(groupnum+1)
            if x < comm_probs(i)
                y = i-1;
                break
```

```
            end
        end
    end

    function prob = prob_assign(g1,g2);
        % We input g1 and g2, which are the groups that two potentially
        % adjacent nodes belong to and return the probability from the
        % array edge_probs.
        if g1==g2
            prob = edge_probs(g1);
        else
            prob = omega_prob;
        end
    end

    % Groups designation is the matrix whose i-th entry is the community
    % number the i-th node belongs to.
    % Prob_mat is the matrix of probability values, where the (i,j)-th
    % entry contains the probability node i and j are connected.
    % M is an upper triangular matrix whose entries are random numbers
from
    % 0 to 1.
    % Lastly, Adjacency_Mat is the functions output, which is the
adjacency
    % matrix for the resulting Stochastic Block Model.

    Groups_designation =
arrayfun(@group_assign,gpuArray.rand(Nodes_num,1));
    Prob_mat = arrayfun(@prob_assign,
Groups_designation,Groups_designation');
    M = triu(gpuArray.rand(Nodes_num,Nodes_num),1);
    Adjacency_Mat = arrayfun(@edge_assign,(M + M'),Prob_mat);


end
```

```
% Stochasitic Block Model for the Worm Network
% Input: N/A
% Output: An adjacency matrix for a graph with an expected
community
% structure and edge density as the Worm network.
% Other functions used: numedges.m
% Last modified May 29, 2015

function Adjacency_Mat = StochasticBlockModelWorm()
    % We start by specifying the number of nodes and communities.

    Nodes_num = 300;
    groupnum = 3;

    % Given the number of groups is groupnum, edge_probs is a
list of
    % length groupnum containing the probability any two edges in
a given
    % community are connected. Omega_prob is the probability two
nodes not
    % in the same community are connected.

    edge_probs = [0.17091,0.16004,0.602632];
    omega_prob = 0.00553761;

    % The comm_probs is a list of length groupnum+1 whose first
and last
    % entries are 0 and 1 respectively. Entry i+1 is the
probability of
    % being in one of the communities from 1 to i. I.e., it is a
    % cummulative probability.

    comm_probs = [0., 0.484746, 0.9322,1];

    % The following functions are defined to be inputted into the
GPU for
    % the purposes of speed.

    function y = edge_assign(x,prob);
        % This function is a simple true/false returner for
parsing to the
        % arrayfun and handled by the GPU.
        if x > 1- prob
            y = 1;
        else
            y = 0;
        end
    end

    function y = group_assign(x);
        % Given the cumulative probability array, comm_probs, we
        % iteratively determine which value range, between 0 and
```

```
1, the
        % random number (also between 0 and 1) belongs to and
returns the
        % index.
        y = 0;
        for i = 2:(groupnum+1)
            if x < comm_probs(i)
                y = i-1;
                break
            end
        end
    end

    function prob = prob_assign(g1,g2);
        % We input g1 and g2, which are the groups that two
potentially
        % adjacent nodes belong to and return the probability
from the
        % array edge_probs.
        if g1==g2
            prob = edge_probs(g1);
        else
            prob = omega_prob;
        end
    end

    % Groups designation is the matrix whose i-th entry is the
community
    % number the i-th node belongs to.
    % Prob_mat is the matrix of probability values, where the
(i,j)-th
    % entry contains the probability node i and j are connected.
    % M is an upper triangular matrix whose entries are random
numbers from
    % 0 to 1.
    % Lastly, Adjacency_Mat is the functions output, which is the
adjacency
    % matrix for the resulting Stochastic Block Model.

    Groups_designation =
arrayfun(@group_assign,gpuArray.rand(Nodes_num,1));
    Prob_mat = arrayfun(@prob_assign,
Groups_designation,Groups_designation');
    M = triu(gpuArray.rand(Nodes_num,Nodes_num),1);
    Adjacency_Mat = arrayfun(@edge_assign,(M + M'),Prob_mat);

end
```

```
function [newA, deg_dist] = PathSelection(A, T,eps)
    N = length(A);
    %Clustering algorithm
    %T = 1000; %Number of iterations for algorithm
    %eps = .5; %Weighting parameter
    % Saving powers of (1 + eps) and (1-eps)

    one_plus_eps_power = (1+eps).^[0:N];
    one_minus_eps_power = (1-eps).^[0:N];

    nodes = randi([1,N],1,T);
    deg_dist = sum(gather(A));

    for n = nodes
          deg_dist = sum(gather(A));
          % deg_dist(n);
         if (deg_dist(n) ~= 0)
            w1 = find(A(n,:));
            w2 = one_minus_eps_power(deg_dist(w1)+1);
            w3 = one_plus_eps_power(deg_dist(w1)+1);
            r= [sum(w2),sum(w3)].*rand(1,2);

            j = 1;
            del = 0;
            while(del == 0)
                r(1) = r(1) - w2(j);
                if r(1) < 0
                    del = w1(j);
                end
                j = j + 1;
            end

            j = 1;
            add = 0;
            while(add == 0)
                r(2) = r(2) - w3(j);
                if r(2) < 0
                    add = w1(j);
                end
                j = j + 1;
            end

            A(n,del) = 0;
            A(del,n) = 0;
            deg_dist(n) = deg_dist(n)-1;
            if n ~= del
                deg_dist(del) = deg_dist(del)-1;
            end
            %M = deg_dist(add);
            nei = find(A(add,:));
            M = length(nei);
            if(M > 0)
                rM = nei(randperm(length(nei)));
% ==========================
```

```
%                   pos = find(A(n,rM),1);
%                   if length(pos)~= 0
%                        A(n,rM(pos)) = 1;
%                        A(rM(pos),n) = 1;
%                        deg_dist(pos) = deg_dist(rM(pos))+1;
%                        deg_dist(n) = deg_dist(n)+1;
%                    else
%                        disp('no edge to add!');
%                   end
% ============================
                iter = 1;
                while A(n,rM(iter)) == 1 && iter < M && M > 0
                    iter = iter + 1;
                end

                if iter <= M
                    A(n,rM(iter)) = 1;
                    A(rM(iter),n) = 1;
                    deg_dist(rM(iter)) = deg_dist(rM(iter))+1;
                    if n ~= rM(iter)
                        deg_dist(n) = deg_dist(n)+1;
                    end
                else
                    disp('no edge to add!');
                end
            end
        end
    end
    newA = A;
end
```